VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Tahová strategická hra pro více hráčů

Turn-based Strategic Game over Internet

2016 Tom Staniček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tom Staniček**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Tahová strategická hra pro více hráčů
Turn-based Strategic Game over Internet

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit tahovou strategickou hru v prostředí Internetu v jazyce Java. Hra spočívá v postupném budování základny a vysílání malých bojových týmu na vybrané mise. Mise budou probíhat tahově. Hra dále umožní vylepšování základny, vytváření bojových jednotek, zlepšování zbraňových systémů, vývoj nových technologií a hru více hráčů.

Hra bude umožňovat:
1. Vytvoření a výstavbu základen.
2. Výstavbu budov/zařízení, vytváření více druhů bojových jednotek, zlepšování zbraňových systémů.
3. Hru více hráčů v prostředí Internetu v kooperativním a kompetitivním režimu.
4. Hráči se budou přihlašovat na mise generované automaticky na serveru.
5. Hru proti jednoduché umělé inteligenci.
6. Program umožní připojení do současně vyvíjeného portálů her v jazyce Java.

Práce bude obsahovat:
1. Implementaci výše popsané strategické hry.
2. Programátorskou dokumentaci řešení s využitím diagramů jazyka UML.
3. Uživatelskou dokumentaci aplikace.

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] DARWIN, Ian F. Java cookbook. 2nd ed. Sebastopol, CA: O'Reilly, c2004, xxiv, 829 p. ISBN 05-960-0701-9. Dostupné z: http://it-ebooks.info/book/2249/

Dále podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

dne 29.4.2016

Poděkování

Děkuji Ing. Davidu Ježkovi PhD. za sdílení svých zkušeností a vědomostí, směrování projektu správným směrem a vstřícnost.

Dále děkuji mému bratrovi Danu Staničkovi za psychickou podporu.

Abstrakt

Cílem této práce je vytvořit počítačovou hru, hratelnou proti ostatním hráčům přes internet. Práce je programovaná v jazyce Java. Jako komunikační prostředek přes internet využívá TCP spojení. Trvalá data jsou ukládána do lokální databáze s využití Java Persistence API (JPA), a také XML soubory. Projekt se skládá ze tří částí. Hlavními částmi jsou server a klient, doplňkovou je editor map.

Uživateli nabízí grafické rozhraní v souladu s futuristickou tématikou hry. Možnost nastavení více uživatelů a herních serverů. Vlastní vytvořené mapy je možné po přihlášení nahrát na server. Uživatel ve hře spravuje vlastní základu, kterou postupně vylepšuje, čímž přináší bonusy pro své týmy. Každý tým se skládá z vojáků a technologií, kterými disponují. Uživatel si navrhuje zbroje i zbraně pro své vojáky v speciálním systému určujícím cenu výzkumu a výroby. Na obrazovce velitelství vidí dostupné mise, náhodně generované serverem z dostupných map a úkolů. V misích je využita výška terénu a tzv. "mlha války".

Klíčová slova

tahová strategie, hra více hráčů, online, vetřelec, člověk, mise, základna, výzkum, editor map, sci-fi

Abstract

Goal of this project is to create the computer game playable with other players over the internet. Project is programmed in Java. As a communication between client and server is uset TCP connection. Permanent data are stored in local databse with usage of Java Persistence API (JPA), and in XML files too. Project have three parts. Main parts are client and server. Additionally there is also map editor.

For users is there graphical user interface in futuristic theme of the game. Users can easily switch between more accounts and servers. Client offers possibility to upload created maps to server. User is managing own base, upgrading it, to provide bonusses for his teams. Every team have own soldiers and technologies. Those technologies are invented by user in special system that is calculating research cost and material cost of invented item. User can sign up on missions in headquarters screen. Missions are randomly generated on server with use of available maps and win conditions. They also uses terrain height and "fog of war".

Keywords

turn-base strategy, multiplayer, online, alien, human, mission, base, research, map editor, sci-fi

Obsah

Seznam použitých zkratek

2D - 2 Dimensional

3D - 3 Dimensional

AI - Artificial Intelligence

API - Application Programmable Interface

DAO - Data Access Object

GIF - Graphics Interchange Format

JPA - Java Persistence API

SHA - Secure Hash Algorithm

TCP - Transmission Control Protocol

XML Extensible Markup Language

Seznam ilustrací

Seznam tabulek

Úvod

Tahové strategie obecně

Tahových strategií se vyskytuje více druhů, může jít také jen o danou část hry. Takovým nejjednodušším příkladem tahové strategie jsou šachy. Jeden hráč provede svůj tah, a po něm pokračuje hráč druhý. Zároveň provedená akce jednoho hráče ovlivní možnosti, které bude mít ve svém tahu protihráč. Tento princip tahů, a vzájemném působení na herní prostředí, se vyskytuje takřka ve všech tahových strategiích.

Tahové strategie se od sebe mohou lišit po estetické stránce i po funkční. Estetická stránka je důležitá hlavně pro zaujetí hráče. Skládá se z grafiky, hudby, příběhu a tématického zařazení. Tématickým zařazením mám zde na mysli například sci-fi, fantasy, 2. světová válka, současnost, a podobně. Příběh může ovlivňovat přímo herní prostředí, a nebo se vyskytovat mimo něj formou videoklipů, textů atd.

Po funkční stránce se mohou tahové strategie lišit v mnoha směrech. Nejznatelnějším rozdílem je, pokud se jedná o počítačovou hru nebo deskovou hru. Po celou práci se však budu zabývat těmi počítačovými, nicméně deskové hry tvoří neméně početnou skupinu tahových strategií, tak jako ty počítačové. Rozdíly ve funkčnosti dále můžeme vidět v omezení počtu akcí za daný tah, v počtu druhů akcí vůbec, v pořadí, v jakém můžeme dané akce vykonávat a různých variabilitách ovlivňujících průběh jednoho tahu. Další rozdíly najdeme v herních objektech, se kterými v průběhu našeho tahu pracujeme. Může jít o bojové jednotky, suroviny, budovy, území, také však o nehmotné schopnosti, výhody, nevýhody a jiné.

Inspirace a zařazení mé práce

Má práce je případ, kdy je tahová strategická část v podobě misí. Před začátkem mise se neprovádí žádné tahy, jde hlavně o správu vojáků a jejich vybavení. Tento systém je inspirován staršími hrami, konkrétně UFO: Enemy Unknown (též X-COM: UFO Defense) z roku 1994 , jeho následovníkem X-COM: Terror from the Deep z roku 1995, taktéž X-COM: Apocalypse z roku 1997 (1). Tyto hry byly vydány firmou MicroProse, a začaly sérii jim podobných her. Ve všech jste měli za úkol chránit svět před útoky mimozemšťanů. K tomu jste stavěli základny, budovu po budově, starali se o finance, prováděli výzkumy, vyráběli zbraně, vybavovali vojáky a letouny, stíhali UFO a odehrávali tahové mise. Jde o rozsáhlé možnosti, kterými jsem se inspiroval při mé práci, a snažil se je poddat vlastním nápadům.

Narozdíl od uvedených titulů se můj projekt zaměřuje hlavně na hru více hráčů, je však možné hrát i proti umělé inteligenci. Ta disponuje stejnými možnostmi jako hráč. Finance jsou nahrazeny materiálem a výzkumnými body. Správa základny je pouze minimální, hlavní zaměření je na samotné týmy. Ty jsou buď cizácké a nebo lidské, a v jedné základny mohou být obojího druhu. Rozdíl se projeví až v samotných misích, kdy proti sobě tyto rasy bojují. Je možné si tak zahrát hru proti všem ostatním týmům, a nebo, společně s týmy stejné rasy, proti opačné rase. Počet vojáků poslaných do mise může být pro každý tým jiný, záleží na počtu startovních pozic na dané mapě. Různé bude i

vybavení ostatních hráčů, protože si je navrhují sami. Projekt obsahuje originální, tužkou kreslené obrázky, dále upravené v grafickém editoru.

Projekt obsahuje několik speciálních částí dostupných hráči. Generátor jmen, nabízený serverem, umí vytvářet náhodná jména po písmenech, také však vybírá z definovaného seznamu jmen ze souboru. V souboru jsou připravena mužská jména z kalendáře.

CreationKit je součástí hry samotné. Jde o univerzální systém pro vytváření věcí ve hře. Momentálně je využíván pouze k vytváření zbraní a zbrojí, je však navržen tak, aby umožňoval jednoduché rozšíření. Jeho úkolem je vypočítat cenu výzkumu, cenu výroby a výsledné vlastnosti tvořené věci podle vstupních parametrů.

Generované mise. Ty, na základě možností vybrané mapy, podporují 3 typy her. Každý typ hry má navíc verzi pro týmovou hru. Tou se myslí rasa proti rase, jinak hraje tým proti týmu. Mapy podporují až 9 týmů o různém počtu vojáků. Typ hry určuje jaké objekty se budou vyskytovat na bojišti a podmínku vítězství. Může ovlivňit však i všechny akce a jejich důsledky. Naimplementovány jsou typy her DeathMatch, CollectResources a DefendExtractor. V DeathMatch je za úkol eliminovat všechny protivníky, DefendExtractor tenhle úkol dále rozšiřuje na ohlídání si vlastního extraktoru dat, a zničit extraktory ostatním hráčům. A v CollectResources je vítěž ten, kdo má na konci hry nasbíráno nejvíce beden s materiálem. Ty jsou rozmístěny na mapě na předdefinovaných pozicích. Herní typy jsou navrženy, tak jako u CreationKit, k snadnému rozšíření.

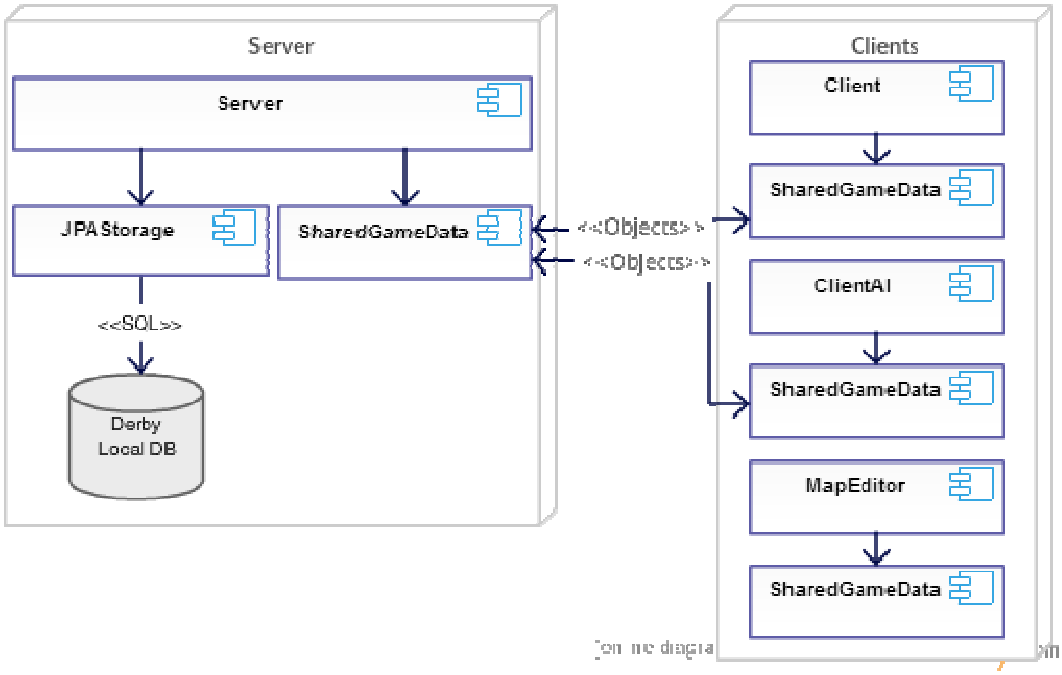
Samotné prostředí misí. Vojáci se v misísch pohybují po různě vysokém terénu. Každý voják má body pohybu, ty jsou ovlivněny pozitivně nebo negativně jeho zbrojí. Každý výškový rozdíl v terénu přidává k ceně pohybu z jednoho pole na druhé. I přes to, že by se dal terén vykreslit pomocí 3D grafiky, ve hře je implementován pouze 2D vykreslováním. Výška je znázorněna tmavostí barvy, a rozdíly jsou vykresleny stínem a barevným přechodem. Každé políčko mapy může mít také různě nadefinovanou barvu, aby nebyl terén pouze šedý. Takzvaná "mlha války" představuje viditelnost svých vojáků. Políčka, která nevidí ani jeden z vojáků hráče, se zobrazí zahalená. Když je na tahu jiný hráč, a provede pohyb nebo střelbu přes odhalené pole, hráč obdrží o tomto zprávu. Klient tyto zprávy načítá postupně, a na začátku tahu je hráči přehraje ve správném pořadí. Nepříjemností při hře více hráčů může být fakt, že hráč, který ztratil všechny vojáky, musí čekat až dohrají ostatní. Což otevírá další možnosti rozšíření, např. odhalit hráči celou mapu, nebo ukončit misis pro hráče předčasně.

Vylepšování základny. Základna má vliv na všechny týmy v ní, ty však musí na její vylepšení posílat vlastní materiály. Vylepšení základny může umožnit vytvoření dalšího týmu, zvýšit limit výzkumu, zeefektivnit recyklaci věcí a nebo zvýšit zisk materiálu z vyhrané mise. Každé vylepšení mají na starost dostupné budovy, momentálně jimi jsou výzkumné oddělení (Research Facility), týmové prostory (Team Quarters), recyklační oddělení (Recyclation Facility) a oddělení materiálů (Material Facility) . Je možné jednoduše přidat další oddělení ovlivňující stejné bonusy. Jelikož každé oddělení je reprezentováno jako mapa, otevírá se možnost implementace stavění mapy základny, která by byla napadnutelná ostatními hráči, či umělou inteligencí.

1 Programátorská dokumentace

1.1 Struktura projektu

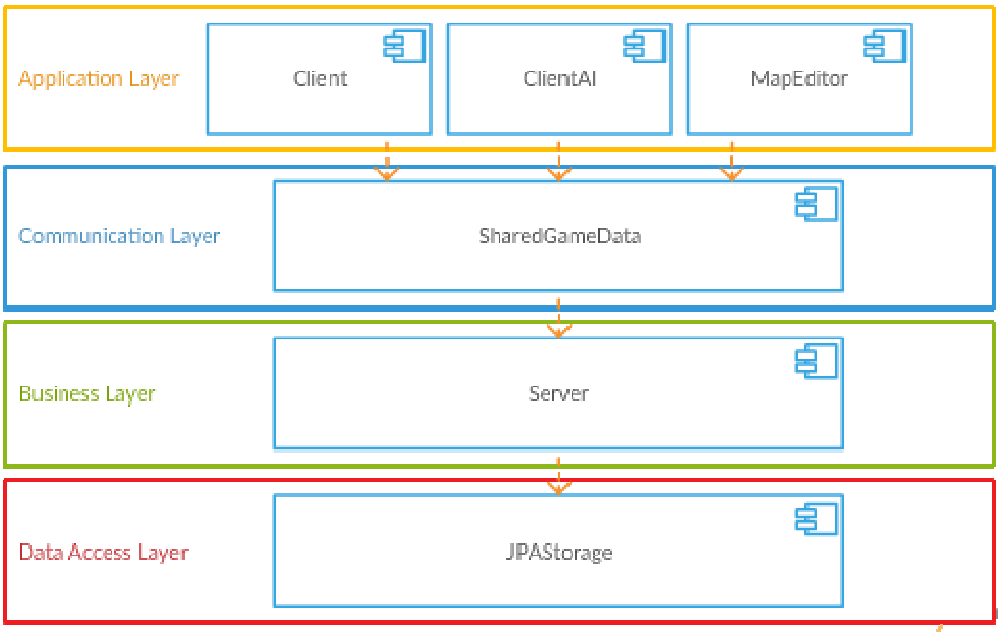
Jelikož jde o projekt, kde komunikuje klient se serverem, rozhodl jsem se rozdělit architekturu celého projektu do 6 modulů. (Všechny užívají stejnou předponu "AvsH_", kterou pro přehlednost neuvádím.) Nejdůležitějším modulem je SharedGameData, který je společný pro server i všechny jeho klienty. Tento modul umožňuje komunikaci se serverem, a navíc usnadňuje práci s mapami. Druhým podpůrným modulem je JPAStorage, který definuje objekty v databázi a přístup k nim. Ten je využíván pouze na straně serveru. Spustitelné moduly jsou zbylé 4, které tvoří Server, Client, ClientAI a MapEditor. Na serveru je implementován herní systém. Client nabízí komunikaci se serverem přes grafické rozhraní. ClientAI má předem nadefinované akce, které má poslat na server. MapEditor slouží k vytváření map, které lze později nahrát na server. Moduly jsou zobrazeny na obrázku Obr. 1.



Obr. 1 Architektura celého projektu

1.1.1 Logické rozdělení vrstev

Moduly lze rodělit do 4 logických vrstev. Aplikační vrstvu tvoří moduly Client, ClientAI a MapEditor. Komunikační vrstu SharedGameData. Logickou vrstvu Server a datovou JPAStorage. Vrstvy mezi sebou komunikují dle obrázku Obr. 2. Toto rozdělení usnadňuje orientaci v projektu, je však třeba zmínit, že SharedGameData neřeší pouze komunikaci, jak je zmíněno v kapitole 1.1 Struktura projektu.



Obr. 2 Logické vrstvy

1.2 Databáze

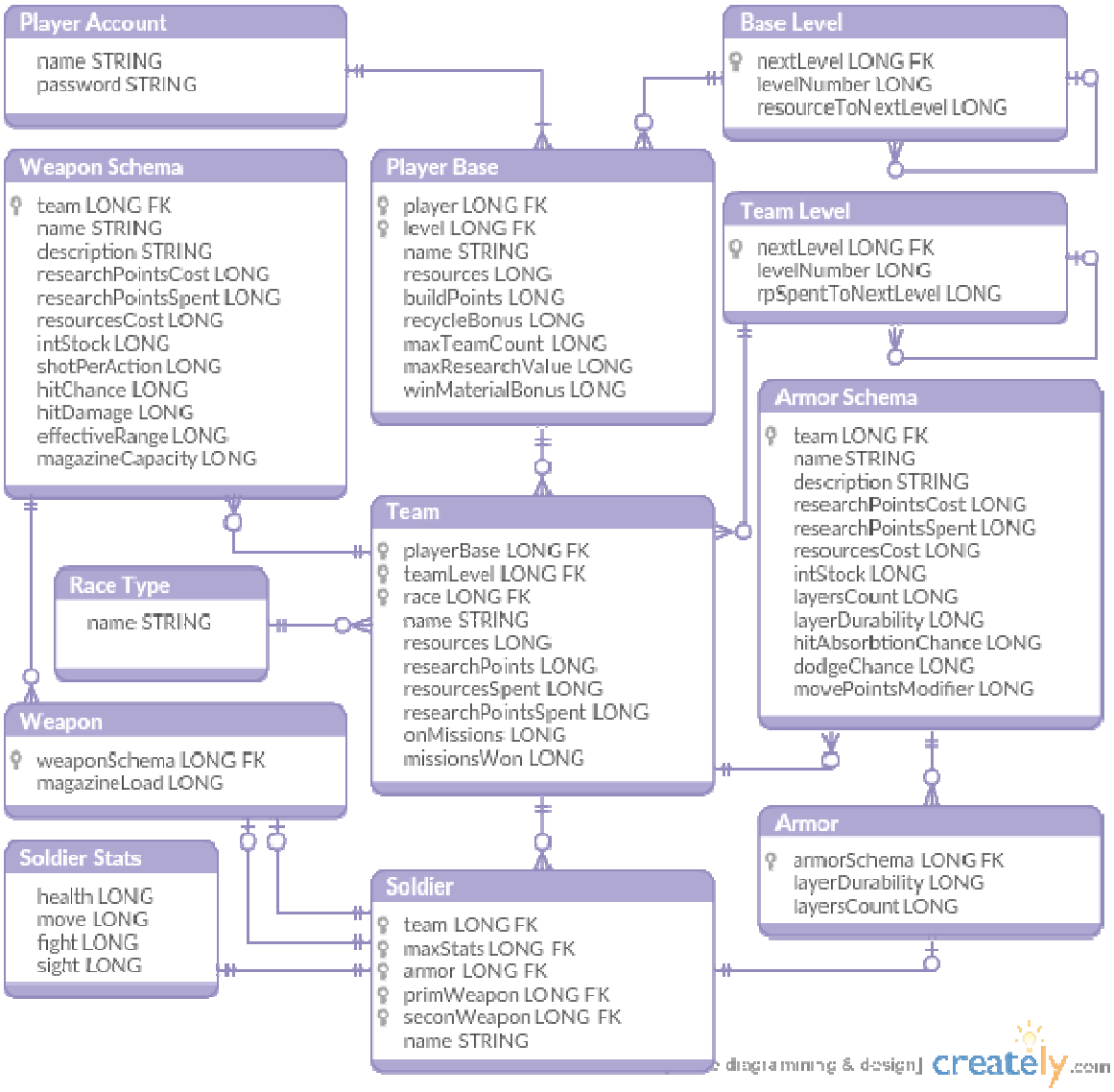
Databázi jsem zvolil jako úložiště trvalých dat, protože je lze jednoduše načítat, ukládat a upravovat. V případě spadnutí programu, databáze nanejvýš ztratí data z právě prováděné transakce. Naopak při ukládání dat do vlastních souborů by bylo vhodné tuto funkcionalitu naimplementovat zvlášť, tvořit zálohy a podobné záležitosti. Do budoucna navíc nabízí toto řešení možnost přístupu k datům i odjinud, než ze serveru. (např. spravování dat z jiné aplikace) . Více přístupů by však znamenalo používat databázi na nějakém serveru, jelikož lokální je závislá na běhu aplikace serveru.

Pro přístup k databázi jsem zvolil Java Persistence API. Persistence provider zajišťuje knihovna eclipselink.jar, jde o spojení s databází a komunikaci s ní. Anotace entit, pro automatické vytvoření tabulek, nabízí knihovna javax.persistence_2.1.0.v201342412130.jar. Lokální databáze je užita z knihovny derby.jar. Tyto 3 knihovny můžete najít v modulu JPAStorage. Všechny jsou navíc volně stažitelné z internetu.

Práce s databází se tak přesunula k definování tříd se správnou volbou anotací a úpravy souboru persistence.xml. Samotná databáze se na základě persistence.xml vytvoří sama, což přináší značné zjednodušení, viz. (2) . Pro komunikaci s ní se v kódu využívá PersistenceManager. Ten jevyužit v DAO objektech.

1.2.1 Entity

Herní entity využívají anotace a společnou rodičovskou třídu GameEntity, implementující rozhraní Serializable a IFormattedPrint (vlastní rozhraní pro podporu výpisu dat entity na řádek) . Relace mezi entitami jsou znázorněny na obrázku Obr. 3.



Obr. 3 Relační diagram

V diagramu jsou záměrně vynechané primární klíče, jenž jsou všechny zděděné z GameEntity, pro přehlednost. U entit ArmorSchema a WeaponSchema není znázorněna jejich společná rodičovská entita ResearchStats, ze které přebírají atributy team, name, description, researchPointsCost, researchPointsSpent, resourcesCost a inStock. Opět pro větší přehlednost. A poslední entita, která na diagramu není znázorněna je SoldierMissionStats, která se neukládá do databáze a je využívána pouze po dobu trvání mise. Dědí ze SoldierStats a je součástí entity Soldier.

PlayerAccount obsahuje přihlašovací údaje. Po registraci server vytvoří hráči základnu (PlayerBase), kterou uloží s účtem do databáze. PlayerAccount sice může mít více základen, aktuální implementace však pracuje pouze se základnou na indexu 0. Jde o další prostor pro budoucí rozšíření.

PlayerBase uchovává údaje o základně. Její úrovně a stavu bonusů. Úrovně jsou vygenerovány při prvním spuštění serveru automaticky, a uloženy v databázi do BaseLevel. Jejich generaci lze ovlivnit v konfiguračním souboru serveru.

Team uchovává statistiky z misí, použitých výzkumných bodů a materiálů. Náleží mu vojáci a výzkumy. Tak jako základna, má svou úroveň znázorněnou tabulkou TeamLevel, která je rovněž vygenerovaná stejným způsobem jako BaseLevel.

TeamLevel a BaseLevel obsahují číslo úrovně. To je pouze reprezentativní. Obojí můžou odkazovat na další úroveň, pokud neodkazují, jedná se o finální úroveň. Pro dosažení nové úrovně u BaseLevel je potřeba dosáhnout daného počtu materiálu na základně. U TeamLevel se úrovně navyšují podle použitých výzkumných bodů.

RaceType momentálně uchovává pouze názvy ras. Rasy jsou načítány při startu serveru, a neměly by se měnit v jeho průběhu. Jiné rasy, než Alien a Human, by v současném stavu způsobily problémy s načítáním obrázků na straně klienta. Jinak by neměly nastat větší problémy v aktuální implementaci. V případě rozšíření ras, by bylo vhodné doplnit o popisek dané rasy, případně umožnit ukládání obrázků týkajících se dané rasy přímo do databáze jako Blob (3).

Soldier obsahuje nejvíce odkazů na ostatní tabulky. Údaje o zdraví, bodech pohybu, akcí a viditelnosti jsou odděleny v další tabulce SoldierStats, a to z důvodu, že SoldierMissionStats uchovává tytéž údaje. Voják mimo misi uchovává pouze maximální údaje, pouze v misích se využívají aktuální hodnoty jeho údajů. Rasa vojáka je totožná s jeho týmem, není tak možné mít vojáky různých ras v jednom týmu, i když by se to zdálo být logické v případě více ras. Pokud by se rasa vojáka ukládala přímo u něj, bylo by třeba upravit týmové verze misí, které s rasami týmů pracují při vyhodnocování výsledků, a balancování přihlášených týmů na danou misi. Pro různorodější tým by bylo řešením vytvoření podras k rasám týmů. Podrasy by by mohly mít rozdílné grafické spracování.

SoldierStats zmíněné v předchozím odstavci by se mohlo zdát nahraditelné konstantníma hodnotama. Tabulka má však svůj účel pro možné budoucí rozšíření. Tím může být náhodné generování statistik pro každého vojáka zvlášť nebo možnost vytvoření genetických vlastností v laboratoři, pomocí CreationKit systému. Hráč by si tak mohl vytvářet geneticky upravené vojáky, případně jiné modifikace.

Weapon a Armor představují vybavené vybavení vojáka. Obojí uchovává aktuální stav dané věci. U zbraně jsou to zbývající náboje v zásobníku, a u zbroje počet vrstev a výdrž nejvyšší vrstvy. Tyto třídy se vytváří po přiřazení vojákovi. Pro odlehčení počtu objektů v databázi, jsou nepoužité věci uloženy pouze ve formě čísla počtu věcí na skladě u příslušného schématu.

WeaponSchema a ArmorSchema jsou rozšířením třídy ResearchStats, jak je zmíněno výše v této kapitole. ResearchStats je přiřazeno k týmu a představuje výzkum. V projektu jsou však použy pouze jeho rozšíření. Výzkumy navíc mohou uchovávat počet již utracených výzkumných bodů, což není aktuálně využito, protože WeaponSchema a ArmorSchema vznikají momentálně pouze využitím CreationKit. Tam se hodnota utracených výzkumných bodů nastaví automaticky na hodnotu potřebných bodů k dokončení výzkumu. Je zde však prostor pro implementaci předdefinovaných výzkumů, na kterých by hráči utráceli výzkumné body. Nyní by docházelo pouze k uchovávání přebytečných dat v databázi ve formě neomezených návrhů zbraní každého týmu, čemuž předchází automatické použití výzkumných bodů na vyvíjenou věc.

V databázi můžeme ještě nalézt entitu Map, která slouží k uchování informací o mapách. Mapy jsou v databázi synchronizovány se složkou s uloženými mapami mimo databázi. Soubor s mapou a její obrázkový náhled se v databázi neukládá. Je to z důvodu větší manuální kontroly nad mapami, a vzhledem k databázi z důvodu velikosti souborů, které se pohybují v řádech desítek až stovek kB. Což je v porovnání s jedním záznamem z jakékoliv jiné tabulky mnohonásobně více.

1.2.2 Data Access Objects

S entitami mohou být spojené různé požadavky na databázi, ty je možno přiřadit do DAO (4), (5) objektu náležícího dané entitě. Zachová se tak přehlednost kódu. U všech DAO využívám vlastní PersistenceWrapper, který zaobaluje samotný PersistenceManager, a ulehčuje práci s ním. Přístup odkudkoliv zajišťují Singleton instance DAOPool a zmíněný PersistenceWrapper.

DAOPool vytváří všechna DAO, specifická pro dané entity, a dá se k nim jednoduše přistupovat. Není tak potřeba řešit vytváření nových instancí pro přístup k datům. Jednotlivá DAO jsou rozšířením abstraktní třídy DefaultDAO, která implementuje základní operace vložení, mazání, upravování a hledání, v databázi. Stačí předat danou třídu entity do generického parametru a konstruktoru. Vlastní operace pouze pro dané entity je třeba naimplementovat zvlášť. Např. vyhledání uživatele s daným jménem a heslem.

V projektu jsou implementovány tyto DAO: ArmorDAO, ArmorSchemaDAO, BaseLevelDAO, MapDAO, PlayerAccountDAO, PlayerBaseDAO, RaceTypeDAO, SoldierDAO, SoldierMissionStatsDAO, SoldierStatsDAO, TeamDAO, TeamLevelDAO, WeaponDAO, WeaponSchemaDAO.

SoldierMissionStatsDAO je přežitkem z původní myšlenky, která počítala s možností ukládat stav mise do databáze. Entita zůstala zachována, tak jsem k ní ponechal i DAO.

1.2.3 Shrnutí databáze

- Vše co se týká databáze najdeme v modulu JPAStorage. Konfigurační soubor persistece.xml taktéž.
- Přístup k databázi zajišťuje DAOPool, případně PersistenceWrapper, obojí singleton třídy.
- Každý herní objekt má zastoupení entitou v databázi. Výjimkou jsou soubory map a stavy vojáků v misích.
- Práci s entitami usnaďnují jednotlivé instance DAO dostupné z DAOPool.
- Automaticky generovanými a spravovanými entitami jsou Map, RaceType, TeamLevel a BaseLevel. Z nichž pouze Map je ovlivnitelná hráči, při nahrání vlastní mapy na server.

1.3 Ostatní uložiště dat

Projekt nepracuje s daty pouze v databázi. Ukládá data několika dalšími způsoby zvolenými vždy podle vhodnosti.

1.3.1 Konfigurační soubor serveru

První soubor, který server potřebuje, je AvsH_Server.txt. Pokud jej server nenajde, vytvoří jej automaticky s defaultním nastavením všech hodnot. Soubor je určen pro konfiguraci serveru, a je uzpůsoben pro přímou editaci a dobrou čitelnost pro člověka. Jde o princip ukládání klíč - hodnota, ve formátu "klíč = hodnota" na řádek. Na prvním řádku se nachází pouze informační text, po prvním řádku se vše ručně načítá striktně v uvedeném formátu. O tohle se stará singleton třída Config, kterou je nutné před použitím inicializovat. Při inicializaci se vytvoří chybějící nebo načte existující konfigurační soubor. Hodnoty jsou ukládány v hashovací tabulce pomocí HashMap. Config obsahuje konstanty pro každý klíč i defaultní hodnotu. Klíče jsou dostupné z vnořené třídy Keys. Config dále nabízí metody pro navrácení hodnoty již převedené na formát čísla pro pohodlný přístup.

1.3.2 Log serveru

Server využívá třídu FileLog pro záznam veškeré komunikace s klienty. Při spuštění a vypnutí serveru se zaznamená do logu čas. Jednotlivé záznamy z komunikace jsou pak zaznamenány pouze IP a portem klienta, typem požadavku a názvem třídy požadavku. FileLog při vytovření vytváří adresář "log", pokud neexistuje, a textový soubor, do kterého zapisuje až do ukončovací zprávy. Název textového souboru je možné změnit modifikací konfiguračního souboru serveru. Poté je však vyžadován restart serveru, aby znovu načetl konfiguraci.

1.3.3 Konfigurační soubor klienta

Klient se po startu pokusí načíst soubor "app-setup.cnfg" s uloženou konfigurací. Pokud soubor neexistuje, vytvoří jej. V něm jsou zaznamenány servery, a k nim náležící přihlašovací jména. Data jsou uživateli nečitelná, uložená serializací třídy SetupData do binárního kódu. Klientská aplikace nabízí grafické rozhraní pro konfiguraci. Záměr takovéto konfigurace spočívá v prvotním nastavení připojení a přihlášení se. Je umožněno nastavení defaultně vybraného serveru a účtu. Uživateli tak stačí nadále již vyplňovat pouze heslo.

1.3.4 Soubory map

Díky MapEditoru je možné vytvářet mapy. Při vytvoření mapy, se automaticky vygeneruje i náhled ve formátu GIF. Samotná mapa je uložená ve formátu XML, v přehledné struktuře. Původně jsem mapy ukládal serializací objektů FileMap, což přinášelo znatelné nevýhody. Takto uložená mapa, po přesunutí třídy FileMap na jiné místo, již nešla načíst. Formát XML tento problém řeší, je navíc možné jednoduše vytvořit další pomocné programy pro editaci map, nezávislé na programovacím jazyce. K vytvoření třídy FileMap ze souboru XML využívám třídu MapXmlUtility. Jelikož třída FileMap neudržuje data příliš přívětivě, je vhodné využívat další třídu MapInformation. Ta využívá MapExtractor, který vytvoří lépe využitelná data o mapě. Nepřívětivost je způsobena předešlým ukládáním serializací, serializace se však nadále používá při přenosu dat mezi klientem a serverem. FileMap se nadále využívá na několika místech v projektu, refaktorizace tak nebude zcela triviální záležitostí, byť užitečnou.

Vytvoření náhledu je možné pouze v editrou map. Využívám k tomu volně stažitelnou, open-source knihovnu freeHEP. Ta dokáže ze Swing komponent vytvořit obrázek daného formátu.

1.4 Server

Server je tvořen stejně pojmenovaným modulem Server, který dále využívá moduly SharedGameData a JPAStorage. Server disponuje konfigurací, jednoduchým uživatelským prostředím, a záznamem do logu. Po nastavení konfigurace, a spuštění, není třeba do chodu serveru nijak zasahovat.

1.4.1 Jádro serveru

Je zastoupeno třídou ServerCore. Běží na samostatném vlákně. Jeho třemi fázemi jsou: inicializace, samotný běh a ukončení. Všchny tyto fáze následují v uvedené posloupnosti. Během inicializace se provedou tyto kroky:

1. Načtení konfigurace třídou Config.
2. Vytvoří a inicializace FileLog pro zpřístupnění logu.
3. Pomocí třídy ResourceLoader se vygenerují příslušné záznamy v databázi, pokud chybí.
4. Vytvoření ServeSocket.
5. Spuštění automatického hledání a odhlašování neaktivních uživatelů třídou ClearIdleClientsTimerTask.
6. Spuštění automatické generace misí v MissionsPool.
7. Načtení vybudovatelných oddělení pomocí inicializace FacilityPool.
8. Inicializace BotsPool, spočívající v přípravě účtů v databázi pro serverem nabízené hráče s AI.

Pokud během inicializace dojde k chybě, jádro se nastaví do stavu "stopped". V takovém stavu se ukončí hlavní smyčka, která následuje hned po inicializaci.

Hlavní smyčka jádra pouze čeká na připojení klientů. Po připojení klienta se vytvoří instance třídy ClientWorker, která je následně spuštěna na samostatném vlákně a uložena do seznamu těchto instancí. Poté se smyčka opakuje, dokud není server uveden do stavu "stopped".

Před bezchybným přechodem do ukončovací fáze se zapíše aktuální čas do logu. Začátkem ukončovací fáze se ukončí spojení se všemi klienty. Poté se ukončí automatická kontrola neaktivních uživatelů a nakonec ukončí zápis do logu. Po této fázi se server nachází ve stavu "exited".

Pro okamžité ukončení jádra je zapotřebí nastavit jeho stav na "stopped", dále pak vytvořit nové spojení, které slouží pouze pro uvolnění hlavní smyčky.

1.4.2 Komunikace s klienty

Komunikaci zajišťuje třída ClientWorker, ta rozšiřuje třídu SocketHandler a implementuje rozhraní Runnable. SocketHandler vytváří dle potřeby nové připojení přes Socket. Využívá se tak TCP připojení. SocketHandler také otevírá ObjectOutputStream a ObjectInputStream pro zápis a čtení dat pomocí serializace objektů. ClientWorker vytvoří pro připojeného klienta instanci ServerService a ServerLoggedService. ServerService implementuje rozhraní IServerService, a ServerLoggedService implementuje IServerLoggedService. ServerService lze využívat jakýmkoliv klientem po připojení k serveru, zatímco ServerLoggedService smí používat pouze přihlášení uživatelé.

ClientWorker po spuštění čeká na příchozí data od klienta. Příchozí data musí implementovat rozhraní IRequest nebo ILoggedRequest, pokud neimplementují, spojení je okamžitě ukončeno. Obojí rozhraní jsou rozšířením rozhraní IReceiverRequest. Tenhle způsob komunikace je rozšířením návrhového vzoru "Command pattern" (6).

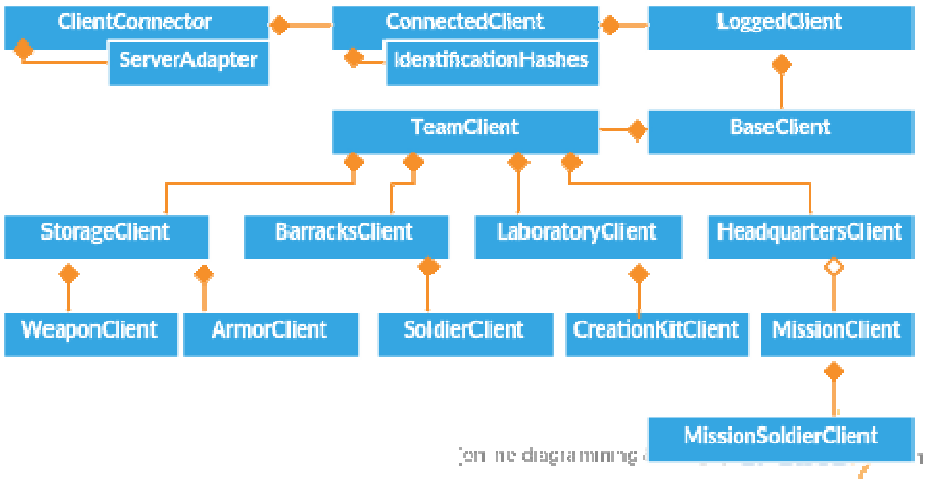
IReceverRequest definuje metodu receive, viz. Obr. 5, ve které se načítají očekávaná data posílána ze serveru, po spuštění metody execute, která je definovaná u rozhraní IRequest a ILoggedRequest. ILoggedRequest navíc definuje metodu getIdentification vracející třídu IdentificationHashes.

Request je abstraktní třída implementující rozhraní IRequest. Dědí z ní RLogin, RRegister a RServerName.

LoggedRequest je abstraktní třída implementující rozhraní ILoggedRequest. Dědí z ní celkem 55 tříd requestů. Na každý požadavek na server existuje právě jedna třída requestu. Dalšími pomocnými abstraktními třídami jsou TeamRequest, SoldierRequest a MissionRequest, které jsou rozšířením LoggedRequest a z nichž dědí velká část ze zmíněných 55 requestů.

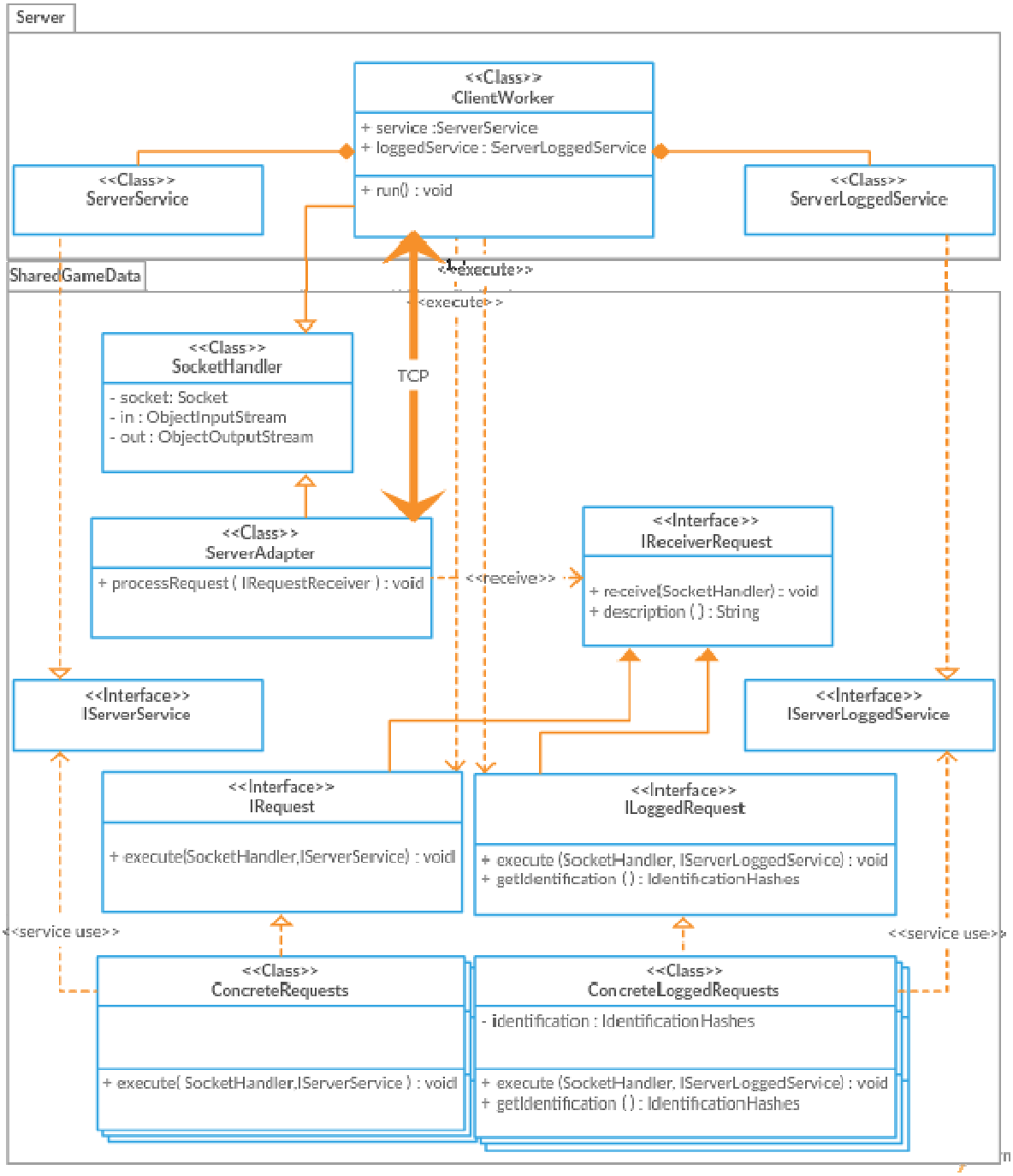
Každý request pracuje s rozhraním IServerService nebo IServerLoggedService, jejichž implementace se nachází na serveru, a je tak klientům zcela skryta. Pro každý požadavek je implementována zvlášť metoda.

Klienti komunikují se serverem pomocí klientských tříd. Ty tvoří hiearchickou strukturu, ve které se z jednoho klienta dá přejít na dalšího. Např. BaseClient po zvolení týmu umožní přístup k TeamClient pro zvolený tým. Navíc, pokud BaseClient nepřistoupí k jinému týmu, v TeamClient zůstane zachován poslední zvolený tým. To přináší výhodu v podobě přímého přístupu k TeamClient bez zadání týmu. Tímhle principem lze procházet celou hiearchii znázorněnou na Obr. 4. Hiearchie začíná třídou ClientConnector, v níž se nachází ServerAdapter. Ten je v konstruktorech předán každé další třídě v hiearchii. Tatkéž všechny třídy následující po ConnectedClient obsahují odkaz na tutéž isntanci IdentificationHashes. U HeadquartesrClienat a MissionClient je záměrně uvedena agregace místo kompozice, protože MissionClient je vytvořený pro každou misi nový.

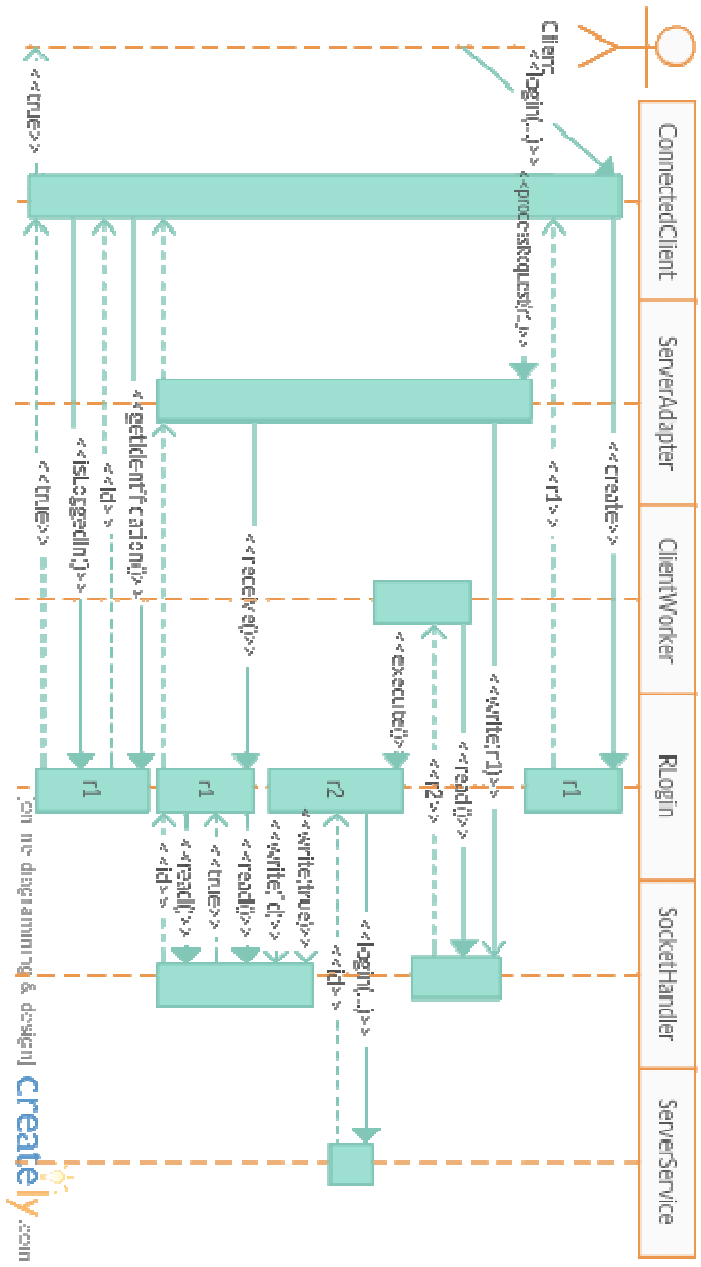


Obr. 4 Hiearchie klientských tříd

Na Obr. 5 je znázorněna komunikace třídy ServerAdapter s třídou ClientWorker. Tu na obou stranách vykonává SocketHandler. Dále lze vidět znázornění volání metody receive u ServerAdapter, což probíhá po odeslání requestu na ClientWorker, ten zase zavolá metodu execute. A právě v ten okamžik se na serveru zavolá daná metoda jedné ze servisních tříd a její výsledek je poslán v metodě execute, a přečten v metodě receive. Po skončení komunikace jsou v daném requestu uložena obdržená data. Celý tento proces je znázorněn sekvenčním diagramem Obr. 6.



Obr. 5 Třídní diagram znázorňující komunikaci se serverem



Obr. 6 Sekvenční diagram úspěšného přihlášení

Na diagramu Obr. 6 lze vidět průběh komunikace s využitím klientské třídy ConnectedClient, která dále využívá ServerAdapter a ten komunikuje přes SocketHandler přímo s ClientWorkerem, který automaticky čeká na příchozí data.

Mimo požadavky ve formě requestů, se mezi klientem a serverem posílají objekty dle návrhového vzoru "Transfer Object". Těchto tříd je vedle requestů taktéž celá řada, jejich počet je však přibližně pouze poloviční. Využívají se jako nosiče dat, a v klientských aplikacích zastupují herní objekty. Mapují a shlukují data ukládaných entit, také však slouží pro různé zprávy z procesů vykonaných serverem.

Speciální případ komunikace nastává během misí. Každý uživatel průběžně načítá seznam instancí zděděných z TransferObject, a ty ukládá pro pozdější zpracování. Toto průběžné načítání provádí pomocná třída MissionManager. Klientská aplikace má dvě možnosti využití této třídy. První je zaregistrování vlastní implementace IMissionRealTimeListener, jejíž metody jsou volány na základě právě načtených těchto tříd: TOPlayerTurn, TOYourTurn, TOMap a TOMissionEnd. To se využívá v implementaci grafického rozhraní misí. Druhou možností je zaregistrování implementace rozhraní IMissionEventProcessor. To obsahuje metodu pro každou událost, kterou může klient z mise obdržet, což zahrnuje i výše uvedené. MissionManager po obdržení TOYourTurn automaticky, pomocí instance třídy MissionEventForwarder, zavolá pro všechny předchozí obdržené objekty dané metody v IMissionEventProcessor ke zpracování.

1.4.3 Bezpečnost

Je řešena na několika úrovních. S první úrovní je možné se setkat v klientské aplikaci. Heslo zadané do přihlašovacího formuláře je z něj obdrženo již jako řetězec zakódovaný pomocí šifrovacího algoritmu SHA-256. Zakódovaný řetězec se pošle na server místo původního zadaného hesla. Ideální by bylo navíc heslo zašifrovat i na serveru, před uložením do databáze.

Dalším bezpečnostím prvkem je samotná komunikace pracující pouze s objekty implemetující rozhraní IRequest a ILoggedRequest. Pokud na server dorazí jakákoliv neznámá třída z výběru requestů, je okamžitě ukončeno spojení s klientem. Navíc samotné implementace servis jsou neznámé pro mobul SharedGameData.

Po přihlášení je uživatel zaregistrován do instance třídy ClientsPool, která uchovává všechny přihlášené uživatele. Z uživatelského jména je vytvořen hash nazván accountHash, a pod tím lze daného uživatele v ClientsPoolu najít. Při každém přihlášení se také vygeneruje unikátní hash, závislý na čase přihlášení, který je přiřazen uživateli po dobu, dokud se neodhlásí. Dynamicky generovaný hash je v projektu nazván identityHash. Uživatel již nadále po přihlášení neposílá své přihlašovací údaje, ale tyto dva hashe, a to při každé komunikaci. Citlivé přihlašovací informace tak putují přes internet pouze jednou. Tohle řešení navíc přináší výhodu v znemožnění připojení se více klientů na jeden účet současně, což je nežádaný stav, který by způsoboval pouze komplikace. Pokud by se pokusil přihlásit znovu, ClientsPool by jej identifikoval podle existujícího accountHash. Po odhlášení je klient odebrán z ClientsPool, a může se poté znovu přihlásit. V případě, kdy klientská aplikace z nějakého důvodu spadně a řádně se neodhlásí by došlo zamezení opětovnému se přihlášení po dobu běhu serveru. Tomu však předchází pravidelná kontrola nečinných uživatelů, kteří jsou poté uvolněni z ClientsPool. Na druhou stranu, aby nedocházelo k odhlašování připojeného, pouze delší dobu nečinného, uživatele, klientská aplikace posíla prázdný požadavek na server v daných časových intervalech. Tím dochází k aktualizací poslední aktivity uživatele, a ten tak není automaticky odhlášen. Počet odhlášených uživatelů je zapisován do logu.

Další bod bezpečnosti se týká práce s databází. Pokud klientský požadavek projde autentizační kontrolou popsanou výše, a jeho požadavek dojde ke zpracování, existuje zde další riziko. Tím je neoprávněný přístup k datům jiného účtu. Při konverzi entity na specifický TransferObject se posílá i její primární klíč. Ten je užit k identifikaci herních objektů. Pokud by byl úmyslně zaměněn za jiný, mohl by uživatel přistupovat k datům ostatních hráčů. Tomu však brání ClientsPool v kombinaci s

ValidAccessService, která obsahuje metody pro kontrolu ID. Pokud je potřeba pracovat s herními entitami, je uživatelská hlavní entita PlayerAccount načtená pomocí identifikačních hashů z ClientsPool. V této entitě se nachází list vlastněných základen, kde je v aktuální implementaci vždy jen jedna. Tím je zabezbečena práce se základnout. Co se týče operací nad týmy, ty mají odkaz na svou základnu jejíž ID musí být totožné se základnou obdrženou z PlayerAccount. Pokud je třeba přistupovat k vojákům, ti zase musí náležet danému týmu. Totéž platí pro přístup k výzkumům. Než se tak provede jakákoliv operace v servisní metodě, musí všechny používané entity projít touto kontrolou prováděnou právě ValidAccessService.

1.4.4 Bezpečnostní rizika

Jedno spočívá v takzvaném "SQL injection", kdy při vyhledávání v databázi pomocí řetězce, je tento řetězec upraven tak, že ukončí dotaz, a za ním pokračuje vlastním dotazem. Tohle hrozí při např. při přihlašování nebo při vyhledávání rasy podle jména. Řešení by poskytovala dodatečná kontrola řetězců, nebo parametrizované SQL dotazy.

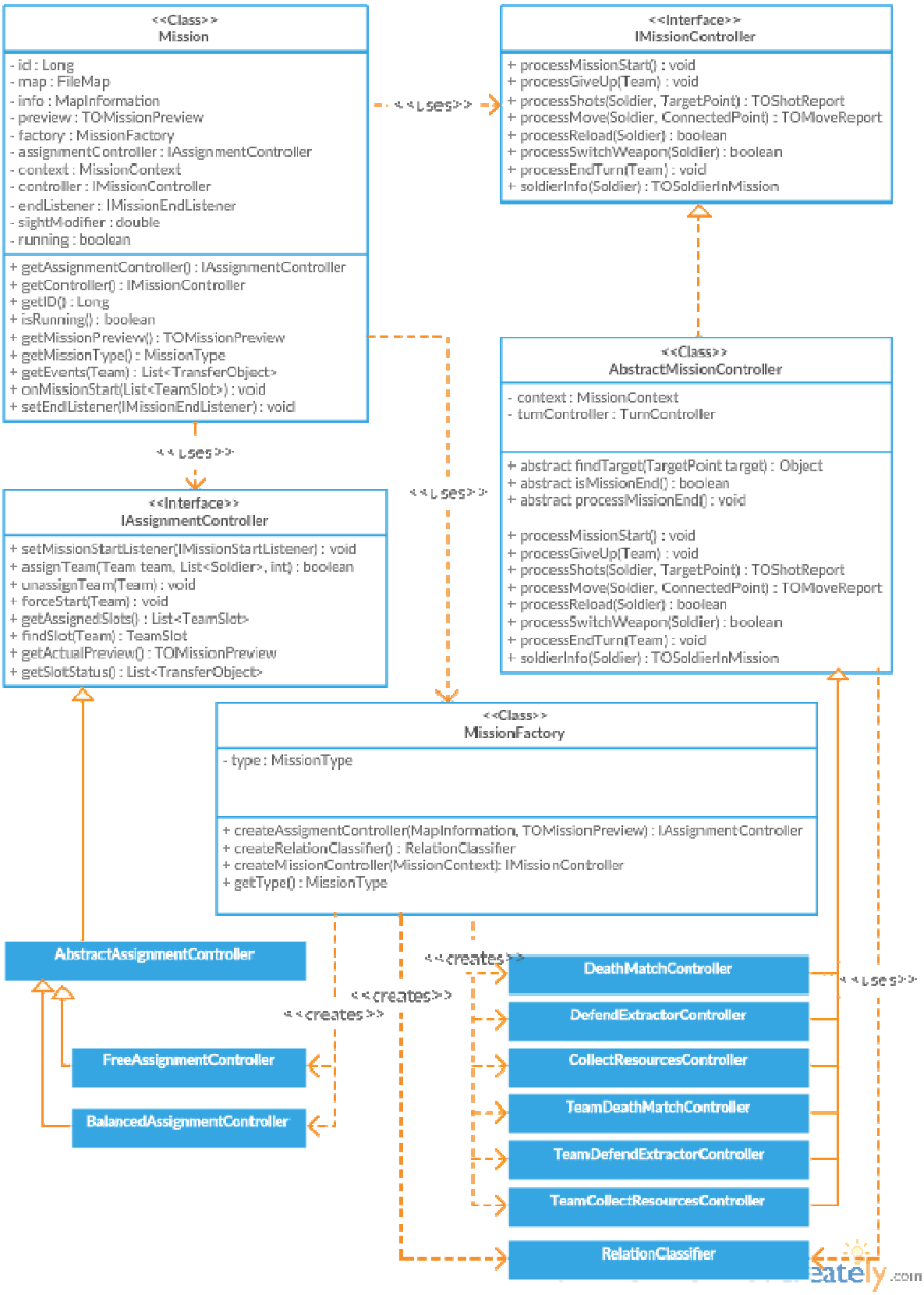
Dalším rizikem tvoří zjištění stále posílaných identifikačních hashů, díky nim by měl útočník však možnost pouze ovlivnit daného až do doby odhlášení. Poté by víc možností, bez znalostí přihlašovacích údajů, neměl.

1.4.5 Generované mise

Mise jsou vytvářeny v singleton instanci MissionPool, pomocí MissionGenerationService každých 5 sekund. V tomto intervalu se zkontroluje počet neběžících misí, a ten je doplněn, až do počtu 5, vygenerováním dalších misí. Běžící mise se klientům v seznamu dostupných misí nezobrazují, ani nejsou serverem posílány.

Generace mise. Pokud při generace dojde opakovaně k chybě, pokus o generování se přesune na další interval, aby nedošlo k zacyklení. Samotný proces vytvoření nové mise spočívá v náhodném výběru mapy z dostupných map registrovaných v databázi. Podle možností mapy, se vybere typ mise MissionType, a tento typ je předán do konstruktoru MissionFactory. Instancí třídy, která dle daného typu vytvoří správnou instanci třídy. Dále se zvýší hodnota ID o 1, která se po dosažení hodnoty 100 počítá zpětně od 0. Pak se náhodně vyberou viditelnostní podmínky pro danou misi ve formě SigntCondition. Jméno se vytvoří z ID, typu mise, mapy a této SightCondition. Informace o mapě shrnuje třída TOMissionPreview, která formou pole bytů, přenáší i obrázek náhledu. Finálním krokem generace je vytvoření instance Mission.

Každá z generátorem vytvotřených instancí hraje ve tříde Mission specifickou roli. Důležitou roli má MissionFactory (viz. Obr. 7) , jelikož řeší tvorbu instancí dle typu hry. Pokud bychom chtěli přidat další herní mód, stačí jej zaregistrovat do MissionType, a v MissionFactory pro něj zvolit vhodný AssignmentController (stará se o průběh přihlašování hráčů na misi) a také chování RelationClassifier. RelationClassifier určuje vztah mezi jednotlivými tými, jde-li o hru "rasa proti rase", vyhodnocuje týmy stejné rasy jako přátelské. Nejdůležitější třídou pro nový typ hry bude spočívat v implementaci IMissionController, kterou lze však usnadnit abstraktní třídou AbstractMissionController, která implementuje všechny společné metody. Implementaci nového módu je tam možné omezit na pouhé definování herních entit v kontruktoru, implementace metody rozhodující o ukončení hry, a vyhodnocení konce hry zahrnujícíc zvolení vítězů a poražených.



Obr. 7 Třídní diagram znázorňující funkci MissionFactory

BalancedAssignmentController se stará o to, jaká rasa se může na misi přihlásit, aby nedošlo k převaze počtu týmů jedné rasy nad druhou. FreeAssignmentController umožní přihlášení všem.

Jednotlivé komponenty tvořící mise jsou propojeny událostmi. Mission poslouchá daný AssignmentController, který rozhoduje, kdy nastal začátek mise. Mission zase rozhoduje, kdy se má mise uvolnit z MissionsPool. Což bývá při klientovi, který si po skončení mise načte jako poslední zbývající události.

Všechna relevantní data mise, se kterými pracuje MissionController, jsou uchovány v MissionContext. Zde najdeme také třídy, které implementují výpočetní algoritmy. Mezi ně patří hlavně SightComputer a MoveController.

1.4.6 Podpora umělé inteligence

Server umožňuje klientským aplikacím povolání hřáče, řízeného umělou inteligencí, který se připojí za tým dané rasy k dané misi, na první volný slot. Implementace tohoto chování se nachází v samostatně spustitelném modulu ClientAI. Pro spuštění je třeba předat parametry v následujícím sledu: IP adresa, port, přihlašovací jméno, heslo, rasa týmu a ID mise. Bot se poté pokusí přihlásit na uvedený server s obdrženými přihlašovacími údaji. Vybere, případně vytvoří tým dané rasy. Zkontroluje, pokud má dostupný alespoň jeden druh zbroje a jeden druh zbraně. Pokud ne, náhodně je vytvoří. Poté se snaží vytvářet vojáky až do počtu 10. A to tak, že najme vojáka, vyrobí pro něj zbroj a zbraň, a přiřadí mu je. Až jsou vojáci připraveni na misi, vyhledá misi podle obdrženého ID, a pokusí se postupně přihlašovat na dostupné sloty, dokud nenarazí na volný slot. Pokud na vonlý slot nenarazí, ukončí se.

Po přihlášení čeká ve smyčce, a načítá události ze serveru v daném intervale. Pokud přijde událost s jinou informací, než o stavu slotů, je to znamení, že začala mise. Posledně obdržené události tak uchová, a předá do instance MissionManager, který je zpracuje. Celý průběh misí je řízen třídou MissionPlayer pomocí vytvoření implementace IMissionEventProcessor, zaregistrované v MissionManager. Aby hlavní vlákno bota neukončilo automatické načítání a provádění událostí MissionManagera, čeká v časované smyčce, a průběžně kontroluje svat mise. Po obdržení události TOMissionEnd se nastaví stav mise na ukončenou. Hlavní vlákno se tak ukončí.

Sever účty botů automaticky vygeneruje v případě potřeby. Přihlašovací údaje každého bota, dostupného na serveru, jsou dostupné z singleton instance BotsPool. Ta funguje na principu návrhového vzoru ObjectPool (7), který dle definice si zaznamená objekt, který poskytl. Poskytnutý objekt neposkytne znovu, dokud není navrácen. Taktéž se chová BotsPool, který si zaznemnává u jednotlivých botů, byli-li vydáni. O navrácení zpátky se stará MissionsPool, který před odtraněním ukončené mise předá její hráče BotsPool, který boty rozpozná a znovu zpřístupní.

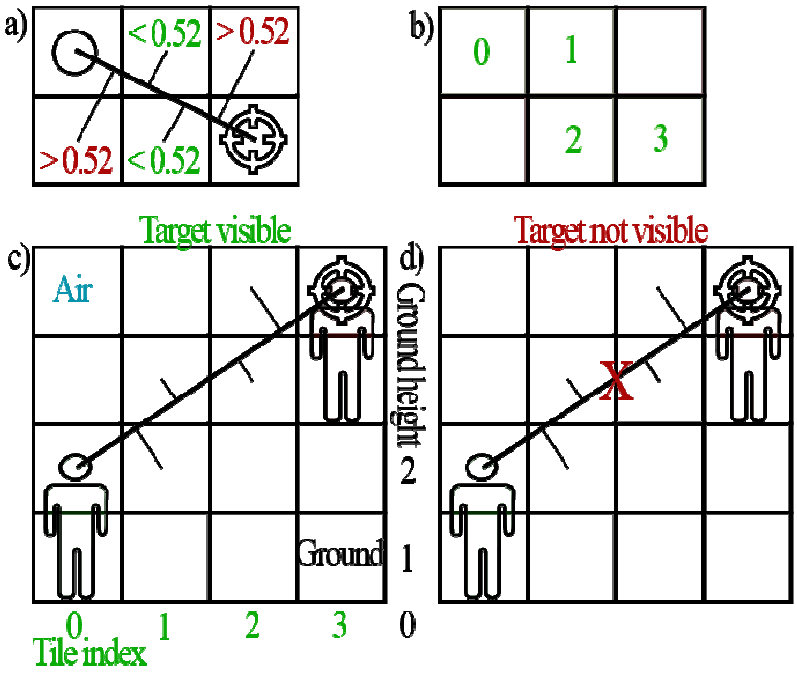
1.5 Výpočetní algoritmy

1.5.1 Výpočet viditelnosti

Jako vtupní parametry jsou poloha a výška zdroje, poloha a výška cíle, a vzdálenost, na kteoru zdroj dohlédne. Zdrojem jsou zde myšleni konkrétně vojáci, a cílem může být jiný voják nebo jakýkoliv jiný nepřátelský objekt. Na Obr. 8 - a) je zdroj označek kruhem, cíl terčem. Algoritmus vypočte vzorec

pro přímku protínající zdroj s cílem. Pro každé pole, které se nachází mezi zdrojem a cílem, se vypočítá nejnižší možná vzdálenost od středu pole k vypočtené přímce. Pokud je tato vzdálenost nižší než koeficient 0,52, dané pole se ve správném pořadí přidá do seznamu kolizních bodů. Ten je znázorněn na Obr. 8 - b). Číslo udává pořadí, červená pole budou z dalšího výpočtu vynechána.

Po zjištění kolizních bodů je dalším krokem sestavení vertikální reprezentace prostoru mapy. Tenhle 2D prostor je široký dle počtu kolizních bodů. Do nichž se počítá i přímo pozice zdroje a cíle. Prostor je vysoký tak, aby pokryl všechny výšky mapy. Ty se mohou pohybovat od hodnoty -10 až po 10. Výška prostoru je navíc ovlivněna i maximem z výšek zdroje a cíle. Pro vojáky je pevně definovaná výška 2. Finální výška prostoru tedy bude 21 + 2. Na obrázcích Obr. 8 -c), d) je pro přehlednost uvedena pouze relevantní výška, týkající se přímo daného výpočtu.



Obr. 8 Algoritmus výpočtu viditelnosti

Po vytvoření vertikálního 2D prostoru, je každý sloupec prostoru inicializován podle výšky dané pozice kolizního bodu na mapě. A to tak, že od nultého indexu se zaznamená úroveň země hodnotou "true". Zbytek prostoru tvoří hodnoty "false" označující vzduch, kterým může dráha viditelnosti procházet. Na tenhle prostor se znovu aplikují kroky pro zjištění kolizních bodů. A nakonec proběhne kontrola, pokud některý z bodů prochází přes zem, pokud ano, není možné na cíl z pozice zdroje dohlédnout.

Jelikož tento algoritmus není zcela triviální, než je spuštěn, je prvně zkontrolována 3D vzdálenost mezi zdrojem a cílem. Pokud je nižší nebo stejná jako dohled zdroje, provádí se zmíněný algoritmus. Algoritmus je implementovaný třídou SightComputer, jež si uchovává informace o terénu mapy. SightComputer je nadále využíván v SightAreaComputer a AimComputer. SightComputer tento algoritmus dále využívá ve výpočtu viditelnosti daného pole vzhledem k více zdrojům. Ty jsou prezentovány instancemi ObserverPoint. Dále tato třída umí rozhodout, zda byla střelba zahlédnuta,

nebo vytvořit z dané trasy pohybu, pouze trasu spatřenou. SightComputer je dále využívám v komponentách MoveController a TurnController.

SightAreaCompuer slouží pro výpočet viditelnosti vojáků. Jde o výpočetně náročný proces, který je však užitečný hlavně v klientské aplikaci. Na serveru se nevyužívá. Algoritmus jednoduše spočívá ve vytvoření 2D pole o rozměrech mapy. Pole obsahuje hodnoty "true" a "false", uvádějící je-li daný bod na mapě viditelný. Umožňuje taktéž k vypočtenému poli provést další výpočet. Tím se dá sestavit pole viditelnosti pro více vojáků. Než jsem došel k tomuto řešení, snažil jsem se náročnost snížit různými postupy. Ty však byly jednak komplikované, a za druhé nezaručovaly věrohodný výsledek, který by byl totožný s otestováním celého pole. Zvolil jsem nakonec tedy zde popsanou možnost. Jediným odlehčením výpočtů je omezení vzdálenosti, na kterou se provádí algoritmus ze SightComputer.

1.5.2 Výpočet šance na zásah

Provádí AimComputer. K výpočtu potřebuje polohu zdroje a cíle, tak jako SightComputer, a efektivní dostřel a šanci na zásah zbraně, pro kterou se výpočet provádí. Prvně je vypočtena 3D vzdálenost mezi zdojem a cílem. Poté je vypočítán faktor pomocí kvadratické rovnice. Faktor značíme "f", vzdálenost "v", a efektivní dostřel "e". Šanci na zásah u dané zbraně označíme "s", a výslednou šanci jako "rs".

$$f = \frac{\left(\frac{v}{e}\right)^2 + 3 \cdot \left(\frac{v}{e}\right)}{4}, rs = s \cdot (2 - f)$$

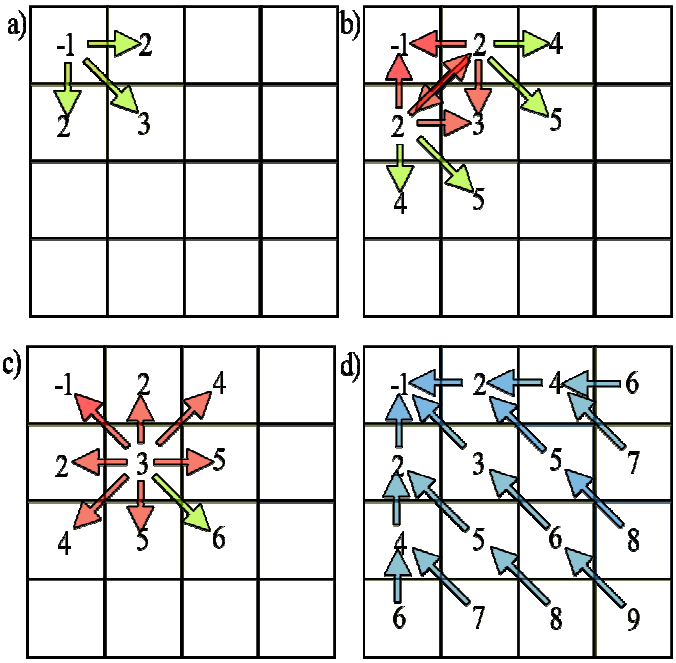
Obr. 9 Rovnice výpočtu šance na zásah

Výsledkem je nezměněná šance na efektivní vzdálenost. Za efektivní vzdáleností šance kvadraticky klesá, opačným směrem naopak přibývá, až do téměř dvojnásobku při minimální vzdálenosti.

1.5.3 Výpočet pohybového pole a cesty k vybranému bodu

Výpočet probíhá ve třídě MoveAreaComputer. Tak jako ostatní algoritmy jsem i tento algoritmus logicky odvodil, a zeefektivnil až do aktuální podoby. Pro inicializaci je potřeba terén mapy, a její rozměry. Algoritmus spočívá v inicializaci pole celých čísel, které dále nazývám pohybové pole. Jsou v něm ukládány ceny pohybu, vypočtené z počáteční pozice. Pohybové pole se na začátku inicializuje na hodnotu 0. Počáteční pozice se zapíše jako -1. Princip spočívá v opakovaném procházení celého pohybového pole. Při každé iteraci, je zadaná výchozí hodnota. Pokud algoritmus narazí na takovou hodnotu, provede z ní výpočet ceny pohybu pro všechny sousedící pozice. Pro každou pozici poté zkontroluje je-li aktuálně vypočtená cena pohybu nižší než hodnota pozice. Pokud ano, hodnota se přepíše na nižší. Po každé iteraci se zvedne výchozí hodnota o 1, a to pokračuje až do dosažení horního limitu. Ten je dán udáním dostupných pohybových bodů, pro které se má výpočet provést.

Na Obr. 10 a) můžeme vidět začátek algoritmu. Pohyb po diagonále stojí 3 body, pohyb vertikálně nebo horizontálně pouze 2 body. Na obrázku je znázorněn výpočet na rovině. Podle rozdílu výšky pozice, na kterou se přechází, je cena vyšší o daný počet bodů, viz. .Obrázek b) ukazuje začátek iterací od výchozího čísla 2. Červené šipky značí pozice, které se nemění. Zelené se přepíšou z inicializační hodnoty na cenu pohybu. Obrázek c) ukazuje následující zpracování iterace po situaci b).



Obr. 10 Algoritmus pohybového pole

V tabulce vidíme přidanou cenu pohybu a výškový rozdíl. Voják se zbojí, která mu ubírá 4 body z pohybu tak není schopen vylézt na pozici o 2 body vyšší.

Výškový rozdíl	Cena pohybu navíc
-4 -	nedosažitelné
-3,-2	2
-1	1
0	0
1	2
2	7
3 +	nedosažitelné

Tab. 1 Tabulka dodatečné ceny pohybu

Prozatím dokáže popsaný algoritmus vypočítat přesně pohybové pole pro jakýkoliv zadaný terén. Pro získání cesty na danou pozici jsem přišel s řešením, které dále upravuje algoritmus. Jediné co přidává je, že při zápisu nebo přepisu hodnoty se na přepisované pozici uloží odkaz, ze které pozice došlo k přepsání. Výsledek těchto cest, vedoucích zpět na počáteční bod, tak můžeme vidět na Obr. 10 c). Nalezení cesty pro jakoukoliv vypočtenou pozici je tak zcela triviální procházení odkazů až na počátek. Světle modrá pole tvoří příklad takové cesty z pozice s modrým polem, a hodnotou 8. Po této jednoduché úpravě pohybové pole již netvoří pouze hodnoty, ale tvořeno instancemi ConnectedPoint. Ty mají implementovány pomocné metody na vytvoření plnohodnotné kopie, na obrácení směru odkazů, a na vypočtení úhlů směrem k navatujícímu bodu. Tohle vše se provede po zavolání metody createMovePath. Pro klienta je tak poměrně jednoduché vygenerovat správnou cestu, kterou je potřeba poslat na server pro provedení pohybu.

1.5.4 Výpočet pomocí grafů

CreationKit je abstraktní třídou speciálně navrženou pro návrh jakékoliv věci s vlastními parametry. Velkou výhodou je její podpora dynamickým grafickým rozhraním na straně klienta. Abychom mohli využívat CreationKit, potřebujeme prvně definovat věc, co se bude tímto nástrojem vytvářet. Tato věc by měla být rozšířením třídy ItemAttributes. V ní je třeba definovat metodu getStats, která vrací list instancí NamedValue, který by měl obsahovat všechny atributy dané věci, které uživatel uvidí.

Dalším krokem je rozšíření CreationKit. Nová třída navíc potřebuje definovat seznam atributů pomocí implementace DynamicAttribute<T>. Generickým parametrem bude dříve definovaná věc. Například ArmorAttributes. Každý atribut, který bude klient nastavovat by měl mít zadaný název a popisek, aby přiblížil uživateli co ovlivňuje. Dále u atributů exitují priority, kdy se aplikují. Atributy s nejniží prioritou se aplikují nejdříve. Tahle prioritizace umožňuje prvně provést inicializaci atributů hodnotami, poté provádět násobení a jiné operace. O postupné aplikování atributů, při vytváření věci, se stará CreationKit.

Popis jednotlivých vlastností atributů:

- hodnota - je libovolně omezena minimem i maximem, je jediným vstupem pro uživatele
- závislosti - odkazují na jiný atribut a mají svou vlastní hodnotu modifikátoru
- modifikátor - je desetinné číslo udávající jakou celkovou váhu daného atributu
- cena výzkumu - váha ceny výzkumu
- cena výroby - váha ceny výroby

Pomocí závislostí mezi atributy máme možnost vytvořit graf, dle kterého se počítá výsledná cena. Při výpočtu ceny, kterou přináší jednotlivý atribut, dochází k mnoha výpočtům.

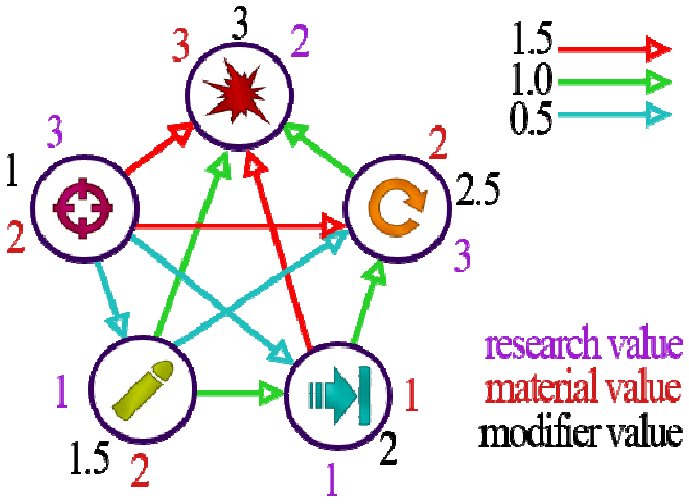
Každá závislost zvedá hodnotu modifikátoru. O kolik se vypočte vynásobením hodnoty atributu závislosti s hodnotou modifikátoru závislosti. To se provede pro každou závislost a vše se přičte k vlastnímu modifikátoru.

Po vypočtení finální hodnoty vlastního modifikátoru se jím vynásobí zadaná hodnota. Pokud chceme zjistit cenu výzkumu, vynásobíme ji dále váhou pro cenu výzkumu. Pro materiál postupujeme stejně.

Celkovou cenu výzkumu zjistíme po sečtení cen ze všech atributů. Totéž pro cenu výroby.

Nevýhodou komplexnosti těchto výpočtů je obtížnost správně vyvážit cenu jednotlivých atributů. Atributy se však dají přehledně graficky zobrazit formou grafu Obr. 11. V bublinách vidíme jednotlivé nastavitelné atributy. Mezi nimi jsou šipkami znázorněny všechny závislosti. Barva šipka udává hodnotu modifikátoru. Čím je hodnota vyšší, tím více tato závislost ovlivní atribut, z něhož šipka vychází. Vlastní modifikátor je u každého atributu znázorněn černou barvou. Cena výzkumu fialovou a cena materiálu červenou.

Toto řešení ze zdá být nejspíš zbytečně složité, jeho úmyslem je však zamezit hráči si vypočítat nejvýhodnější kombinaci atributů dané věci. Pokud by byl výpočet příliš jednoduchý, hráč by nacházel potřebu v přepočtech. Takto bude prostě zkoušet různé kombinace a využije plně možnosti CreationKit.



Obr. 11 Graf při výpočtu ceny zbraní

1.5 Klient

Hlavní třída AvsHClient obsahuje statické instance tříd GUIFactory, ClientConnector a SetupData. Za zmínku stojí instance singleton třídy ApplicationWindow, které tvoří hlavní okno klienta. Podporuje režim na celou obrazovku i režim oken. Mezi režimy lze odkudkoliv přecházet pomocí klávesy F12. Celá klientská aplikace je v podstatě grafickým rozhraním pro práci s klientskou hiearchií komunikující se serverem.

1.5.1 Grafické rozhraní

Klientské grafické rozhraní využívá pouze knihovny Swing. Téměř každá komponenta knihovny je předělaná tak, aby podporovala průhledné pozadí. V některých případech bylo k dosažení průhlednosti potřeba rozšířit a upravit více tříd. Například CustomScrollbarUI pro posuvníky v ScrollPane. Ty jsou vykreslovány z části ručně.

Pro tvorbu komponent je k dispozici třída GUIFactory, inspirována navrhovým vzorem "Factory".

1.5.2 Vykreslování mapy

Každé políčko mapy je tvořeno jednou komponentou MapTile, která dědí z JComponent knihovny Swing. MapTile přepisuje vykreslovací metodu paintComponent. Vykreslování je dokonce závislé na okolních MapTile, podle kterých se na komponentě vykresluje stín, případně barevný přechod. Všechno tohle vykreslování můžeme najít v kódu MapTile. Pro odlehčení složitosti jsem vytvořil pomocnou třídu TilePaintProperties, která uchovává všechna data s vykreslováním spojená. Data v TilePaintProperties jsou rozdělena na dvě skupiny, jedna je určena k nastavování a druhá se pro dané nastavení dopočítává. Je umožněno nastavení poměru velikosti barevného přechodu vzhledem k velikosti políčka. Lze nastavit velikost políčka. Maximální a minimální černobílý odstín, z nějž se dopočte konečná barva dle výšky terénu daného políčka. Vykreslování barevného přechodu se provádí pro výškový rozdíl jednoho bodu, pokud je rozdíl větší, vykreslí se ostrý stín. MapTile dále umožňuje vykreslit informaci o výšce, nebo speciální znak. Tím jsou znázorněny např. startovní pozice vojáků.

Všechny MapTile jsou spravovány pomocí MapView. Té je možno přiřadit MouseAdapter, který se automaticky registruje do každého přidaného políčka. Před použitím MapView je nutné nastavit TilePaintProperties pomocí metody setAutomaticTilePaintProperties. Při přidání políčka se tak jeho nastavený nastaví podle jednoho globálního. Metoda refreshPaintProperties tak poté slouží k opětovnému nastavení. Při změně výšky jednoho políčka není potřeba překreslovat celou mapu, ale stačí využít metody, která najde okolní políčka a překreslí pouze ty.

1.5.3 Vykreslování misí

Mise využívají komponentu LayeredPane. Na její nejnižší vrstvě se nachází MapView vykreslující mapu. Ve vrstvě nad ní se nachází IneractiveGrid, což je třída která užívá několik pomocných vykreslovacích a nimačních tříd. IneractiveGrid je rozšířením JPanelu, a využívá nastavení dvojího vykreslování. I přesto však není příliš optimalizovaná, aby nedocházelo k občasnému přeblikávání. Mezi využitými třídami jsou FOWPainter, MovePainter, CursorPainter, UnitPainter, ShotAnimator a MoveAnimator.

FOWPainter slouží pro vykreslení "Fog Of War", takzvané mlhy války. Má přímý odkaz na pole viditelnosti, se kterými pracuje SightAreaComputer. Dle tohoto pole vykresluje mlhu na všechna neodhalená políčka.

MovePainter slouží k vykreslení pohybového pole, které používá přímo od MoveAreaComputer. Mimo pohybové pole vykresluje i cestu pohybu z počátku na dané políčko.

CursorPainter umí vykreslit čtercový kurzor a zaměřovač s hodnotou šance na zásah.

UnitPainter slouží pro vykreslování jednotek ve formátu TargetPoint. Vykreslí je v daném úhlu natočení. Navíc pod obrázky vykresluje barvy podle TargetClassification, aby šlo rozeznat nepřátelé od přátelských a vlastních jednotek. A poslední věcí, kterou umí, je vykreslení označení dané jednotky.

ShotAnimator vykresluje střelbu v podobě laseru. Je rozšířením třídy AbstractAnimator, tak jako MoveAnimator, který sice nevykresluje nic, také provádí kroky animace v daných intervalech. Při každém kroku je IneractiveGrid překreslována, jelikož se do ní vše vykresluje.

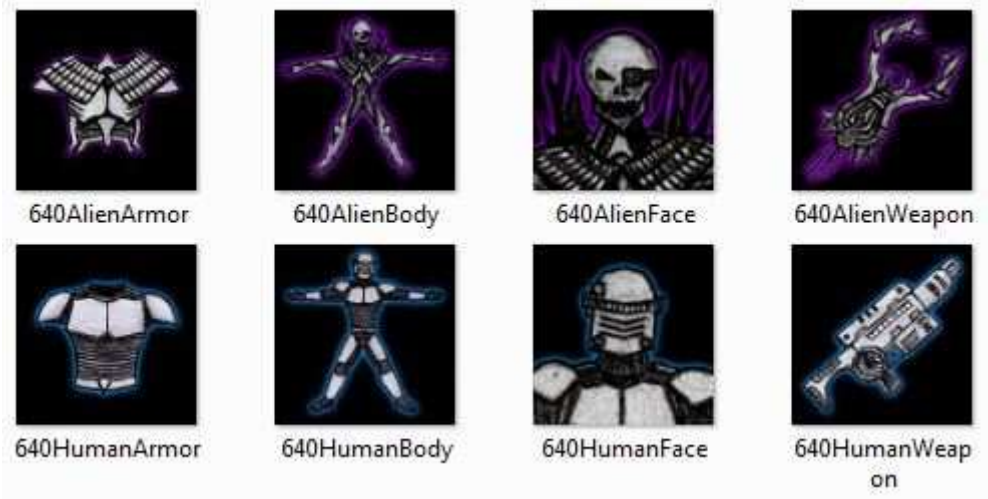
IneractiveGrid dále umožňuje vyvolat událost po kliknutí na vlastního vojáka, nepřátelského a na pohybové pole. Klient v těchto případech odesílá požadavek na server a ihned přehraje výsledek pohybu nebo střelby. V případě označení jde pouze o akci na straně klienta.

1.5.4 Vlastní grafika

Veškeré obrázky, které lze najít v projektu jsou mým výtvorem. Většina je kreslená tužkou, oskenováná a dále upravena. Jako grafický editor jsem použil Photoshop verze CS3, se kterým mám dlouhodobé zkušenosti. Práce v něm je tak spíše odpočinkovou aktivitou. Vznikly tak například 2 verze loga, viz. Obr. 15. K uvedeným obrázkům na Obr. 12, Obr. 13 a Obr. 14 lze přistupomat pomocí statických metod třídy Images. Images navíc obrázky shlukuje jako "enumeration"/výčet, přes který je možné obrázek načíst nejjednodušeji.

Obr. 12 Obrázky užité v misích

Obr. 13 Soubor ikon



Obr. 14 Vybavení cizáků a lidí

Obr. 15 Nové logo v porovnání se starším

1.6 Editor map

Přináší hlavně základní nástroje pro výtváření map. Samotná implementace se nachází pouze v jedné třídě, a není v příliš přehledná, jelikož se zde nachází v podstatě vše. Odlehčením je však využití podpůrných tříd pro práci s mapama z modulu SharedGameData. Tyto třídy byly původně v modulu MapEditor. Rozhodl jsem je však přesunout z důvodu pozdějšího využití v klientské aplikaci.

V editoru je možné terén pohodlně kreslit, obarvovat a vkládat herní prvky. Těmi jsou startovní pozice, vlajky sloužící jako pozice extraktoru, a bedny. Mezní hodnoty mapy lze taktéž upravovat dle potřeb tvůrce. Omezením, které přináší odstíny políčka dle jeho výšky, je barevný rozsah 0-255. Ten pro zobrazení 20 úrovní již začíná splývat, a stává se těžce rozeznatelný. To není dobré vzhledem k orientaci hráče na takové mapě.

Závěr

Práce byla velmi náročná a obsáhlá. Zjistil jsem, že vzhledem k termínu odevzdání, není nejlepším počínáním se snažit do práce implementovat spoustu svých nápadů kolem daného tématu. Mnohem jednodušší by bylo se striktně držet zadání a práci zbytečně nekomplikovat. Což mi bohužel moc nevyšlo, vzhledem k zálibě ve hrách tohoto typu, a obecně vymýšlení promyšlených herních systémů, které se snažím uplatnit většinou v deskových hrách. S výsledkem jsem však spokojený, a většina mých nápadů funguje tak, jak jsem je zamýšlel. Práce s poli, tvořícími mapy, mě bavila i v předešlém školním projektu. Zde jsem však namísto náhodně generované mapy řešil zcela jiné problémy. Největším problémem bylo vymýšlení algoritmu viditelnosti. Zkoušel jsem spoustu variant, jak obejít výpočet viditelnosti pro každé pole, nikam jsem se však nedostal, než k původní úvaze, jelikož všechny navržené algoritmy byly příliš nedokonalé a nerealistické. I aktuální algoritmus má svá omezení. Jde konkrétně o vytvoření vertikální mapy, ta by správně měla mít různé délky sloupců dle úhlu přímky. I přes to je však výsledek přesvědčivý. Dále bych chtěl projít jednotlivé body zadání.

Shrnutí bodů zadání

1. Výstavbu základny bych jistě mohl více propracovat. Bohužel mi na to již nezbyl čas, tak je implementována v minimálním rozsahu.
2. Výstavba budov/zařízení uvedeno v druhém bodu souvisí v podstatě s prnvím bodem zadání. Druhy bojových jednotek se dá chápat jako různě vybavené jednotky. Tyto body jsou však celkem obecně napsány, konkrétní implementace záležela na vlastním uvážení.
3. Hra přímo staví na hře více hráčů. Hráči mezi sebou zápasí v misích a testují své vyzkoumané zbraně a zbroje.
4. Generování map je implementováno nad vlastní očekávání dobře.
5. I velmi jednoduchá umělá inteligence oživila testování misí. Není však momentálně žádnou výzvou, pokud se nehraje vyloženě hloupě. Záměr u umělé inteligence však byl ve správném rozložení struktury projektu. Se strukturou jsem nanejvýš spokojen, a také s klientskou hiearchií, jejíž užití je velmi jednoduché. Klient se dobře implementoval jak do grafického rozhraní, tak při naskriptovaném, pevně daném průběhu u umělé inteligence.
6. Herní portál bohužel nebyl v hotovém stavu, tento bod je tak v projektu zcela vynechán.

Vlastní přínos

Přínosem mi byl projekt v mnoha směrech. Naučil jsem se pracovat s verzovacím systémem GIT. Zdokonalil jsem se v práci s velkými projekty, jejich rozvžení do přehledné struktury. Lépe chápu samotný programovací jazyk. Velkým přínosem bylo také zjištění, že ObjectOutputStream a ObjectInputStream si spravují odesílaná data na první pohled téměř nezjištným způsobem, který když ten, kdo je využívá nezjistí, potýká se spoustou zvláštních, těžce vysvětlitelných chyb. Ověřil jsem si také důležitost návrhových vzorů, které dokáží celou problematiku zjednodušit.

Možné další rozšíření projektu

- Přidání více módů výzkumu zbraní a zbojí pro CreationKit, implementace genetických úprav vojáků.
- Více typů misí a rozšíření ukládáných dat o jednotlivých mapách.
- Vykreslení misí ve 3D, spolu s 3D modely herních objektů.
- Implementace ochrany proti SQL injection.
- Dodatečné šifrování hesel před uložením do databáze.
- Umožnit ukládání stavu mise do databáze.
- Umožnit přímé vytváření mapy základny pomocí výstavby budov. S tím spojená možnost útoku na základnu jiného hráče. A také náhodné události, při kterých je základna napadena hráčem ovládaným umělou inteligencí.
- Zlepšit animace v misích.
- Přidat hudbu na pozadí a zvuky do hry.
- Vojákům umožnit postupně zlepšovat své statistiky.
- Sada výzkumů mimo CreationKit.
- Rozšíření editoru mapu v ohledu na hromadnou editaci větší plochy.
- Nastavování vzhledu grafického rozhraní
- Optimalizace vykreslování mapy a misí

Literatura

1. X-COM. *Wikipedia.* [Online] 25. únor 2016. [Citace: 23. duben 2016.] https://cs.wikipedia.org/wiki/X-COM.

2. **O'Brien, F. Michael.** EclipseLink/Examples/JPA/Derby. *wiki.eclipse.org.* [Online] 27. leden 2011. [Citace: 27. duben 2016.] https://wiki.eclipse.org/EclipseLink/Examples/JPA/Derby#Embedded_Derby_Driver_VS_Derby_Client.

3. **Morris, Stephen B.** Five Steps to Managing Unstructured Data with Derby. *www.ibmpressbooks.com.* [Online] 14. Prosinec 2007. [Citace: 16. leden 2016.] http://www.ibmpressbooks.com/articles/article.asp?p=1146304&seqNum=3.

4. **Fowler, Martin.** Catalog of Patterns of Enterprise Application Architecture. *http://martinfowler.com/.* [Online] leden 2003. [Citace: 12. listopad 2015.] http://martinfowler.com/eaaCatalog/.

5. **oodesign.** Design Patterns. *OODesign.com.* [Online] [Citace: 3. říjen 2016.] http://www.oodesign.com/.

6. **tutorialspoint.** Design Patterns - Command Pattern. *tutorialspoint.* [Online] [Citace: 25. duben 2016.] http://www.tutorialspoint.com/design_pattern/command_pattern.htm.

7. **Alexander Shvets, Gerhard Frey, Marina Pavlova.** Object Pool Design Pattern. *SourceMaking.* [Online] 2006. [Citace: 6. listopad 2015.] https://sourcemaking.com/design_patterns/object_pool.

8. JPA Entity Fields. *www.objectdb.com.* [Online] [Citace: 5. duben 2016.] http://www.objectdb.com/java/jpa/entity/fields.

Přílohy

I. Zdrojové kódy všech modulů - příloha na CD.

II. Použité obrázky - příloha na CD

III. Uživatelský manuál - příloha na CD.